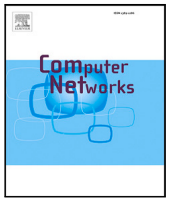
Computer Networks 216 (2022) 109214

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



IntOpt: In-band Network Telemetry optimization framework to monitor network slices using P4☆

Deval Bhamare [a], Andreas Kassler [b], Jonathan Vestin [b], Mohammad Ali Khoshkholghi [c,*], Javid Taheri [b], Toktam Mahmoodi [c], Peter Öhlén [d], Calin Curescu [d]

[a] Department of Computer Science, University of Surrey, UK
[b] Department of Computer Science, Karlstad University, Sweden
[c] Department of Engineering, King's College London, UK
[d] Ericsson Research, Stockholm, Sweden

ARTICLE INFO

Keywords:
In-band Network Telemetry (INT)
Monitoring
P4
Network function virtualization (NFV)
Service function chain (SFC)

ABSTRACT

The emergence of Network Functions Virtualization (NFV) is being heralded as an enabler of the recent technologies such as 5G/6G, IoT and heterogeneous networks. Existing NFV monitoring frameworks either do not have the capabilities to express the range of telemetry items needed to perform management or do not scale to large traffic volumes and rates. We present IntOpt, a scalable and expressive telemetry system designed for flexible NFV monitoring using active probing and P4. IntOpt allows us to specify monitoring requirements for individual service chain, which are mapped to telemetry item collection jobs that fetch the required telemetry items from P4 programmable data-plane elements. We propose mixed integer linear program (MILP) as well as a simulated annealing based random greedy (SARG) meta-heuristic approach to minimize the overhead due to active probing and collection of telemetry items. Using P4-FPGA, we benchmark the overhead for telemetry collection. Our numerical evaluation shows that the proposed approach can reduce monitoring overheads by 39% and monitoring delays by 57%. Such optimization may as well enable existing expressive monitoring frameworks to scale for larger real-time networks.

1. Introduction

Recently, various network functions such as domain name servers (DNS), deep packet inspectors (DPI) as well as mobile core functions such as Mobility Management Entity (MME), Home Subscriber Server (HSS), Serving Gateway (SGW), Evolved Packet Data Gateway (ePDG) and others are being deployed as virtual network functions (VNFs), or simply virtual functions (VFs). Instead of being built on dedicated hardware, network functions are now being implemented as software at application layer that run on top of general purpose hardware through virtualization, called as, virtualized network functions (VNFs). These VNFs are interconnected to form service function chains (SFCs) [1–3]. As different SFCs coexist on the physical network belonging to different tenants, it has led to another novel networking paradigm called as network slicing (NS). NS allows to build a set of dedicated networks, each adapted to serve one type of business customer. These dedicated networks would permit the implementation of tailor-made functionality, such as 5G/6G, IoT, and heterogeneous network operations specific to needs of every customer [4,5].

The service flows (SFs) deployed over such network slices have stringent service level agreements (SLAs) including end-to-end latency, throughput, buffer sizes among others. To meet the SLAs and measure the quality of service (QoS), network operators are mandated to constantly measure the network state. The parameters supported by P4 framework and which need to be monitored include (1) queue buffer sizes, (2) path followed by packets (arrival and departure ports at each hop), (3) switch ID, (4) ingress and egress timestamps, (5) available bandwidth, (6) queue delays, (7) packet arrival rate, (8) throughput and others.

Managing and scaling service flows (SFs) in network slices require the collection and analysis of network statistics and states in real time.

☆ This work is an extension of our previous work "IntOpt: In-band Network Telemetry Optimization for NFV Service Chain Monitoring" presented in IEEE ICC 2019 (Bhamare et al. 2019). The work has been significantly extended (approximately 50%) to be presented as a novel work to the current journal.

* Corresponding author.
E-mail addresses: d.bhamare@surrey.ac.uk (D. Bhamare), andreas.kassler@kau.se (A. Kassler), jonathan.vestin@kau.se (J. Vestin), ali.khoshkholghi@kcl.ac.uk (M.A. Khoshkholghi), javid.taheri@kau.se (J. Taheri), toktam.mahmoodi@kcl.ac.uk (T. Mahmoodi), peter.ohlen@ericsson.com (P. Öhlén), calin.curescu@ericsson.com (C. Curescu).

https://doi.org/10.1016/j.comnet.2022.109214
Received 10 March 2022; Received in revised form 13 July 2022; Accepted 21 July 2022
Available online 3 August 2022

Measuring network parameters help the operators to perform quality control, congestion control, anomaly detection, capacity planning in data centers as well as backbone networks [6]. Measurements are important when network state changes frequently due to the changes in the traffic flowing through the network, addition or deletion of the services as well as changes in the underlying physical topology. Many factors contribute to the changes in the physical network, one of them and the most common being link congestion or failures. Also, with the advent of network function virtualization (NFV) [7] and service function chaining (SFC) [8], network services have become dynamic, changing their states quite frequently [9,10].

Network monitoring and measurement tasks demand resources including network bandwidth, computational resources as well as memory. Introducing monitoring flows (MFs) for network monitoring in non-optimized manner may result in additional overhead, affecting the actual traffic and performance of the service function chains (SFCs), which are the source of the revenue for network operators. Existing network measurement solutions mainly focus on a good trade-off between expressiveness, accuracy, speed and scalability [11,12]. Although existing works have made great contributions towards network monitoring and measurement, they do not focus on one fundamental need, that is, not overwhelming the network with monitoring overheads while being expressive as well as scalable. For example, systems such as Chimera [13] and NetQRE [14] can support a wide range of queries using stream processors running on general-purpose CPUs, but they incur substantial bandwidth and processing costs to do so and hence are not scalable.

In this work, our contribution is a proposal of "IntOpt", a scalable, yet expressive active telemetry collection framework, targeted at monitoring a set of service function chains which are deployed over a substrate network. In our hybrid approach, the IntOpt controller determines the set of optimal monitoring flows (MFs) which minimize the total overhead of the network monitoring [1]. The framework then determines the respective set of INT-sources, sinks and forwarding nodes to collect all required monitoring items from the switches in the substrate network on which SFCs are routed. The proposed framework uses active telemetry probing [15], that is, inserting separate monitoring probes for the MFs periodically in the network to gather telemetry information from the data-plane using programmable data plane elements and P4. The sink nodes in the physical network send the probes along with the collected telemetry information back to the controller for further analysis. The IntOpt controller executes its commands by communicating them through the SDN controller, which in turn communicates with the underlying physical switches through the control plane [16].

The rest of the paper is organized as follows. In Section 2 we discuss the related work in brief. The IntOpt architecture is explained in Section 3. In Section 4, we have developed mixed integer linear programming (MILP) model to formulate the problem of preparing the monitoring flows so that all service flows are adequately monitored. Since MILP can solve the problems with limited size due to its computational complexity, we have also implemented simulated annealing based random greedy (SARG) meta-heuristic approach in Section 5. For input to our meta-heuristic, we benchmark the P4 INT framework using P4FPGA [17] to approximate the delay induced by INT-operations in Section 6. We then compare the performance of proposed SARG approach against the optimal solution obtained by MILP model. Also, we compare the performance gain obtained for larger topology and bigger problem size by comparing the SARG approach with a naïve approach which is unaware of the optimization policies and used for the deployment of unplanned monitoring flows. Finally, Section 7 concludes the paper addressing future directions. Full forms of the acronyms used in this paper are given in the Table 6, at the end of the article.

2. Related work

Network monitoring has been an area of interest for many researchers and organizations for a long time [18,19]. Standards such as ITU-T Recommendation Y.17312007 [20] and IEEE 802.1ag-2007 [21] have been proposed already to define protocols and practices for OAM (Operations, Administration, and Maintenance) for network paths through 802.1 bridges and local area networks (LANs). Such standards are Ethernet compliant, however, they are not developed to cater the needs of recent networking paradigms such as network function virtualization (NFV) and their stringent service level agreements (SLAs). These protocols are difficult to update according to the dynamic demands of NFV and SFC paradigms and hence are not effective [22,23].

Protocols such as Simple Network Management Protocol (SNMP) [24] have been in existence for long, which uses polling mechanism to fetch the data stored in management information base (MIB). Such protocols are, however, ineffective, especially because they are not fast enough [11]. For example, with the recent advancements such as NFV, network states change quite rapidly and frequently. Due to polling, significant event or information, such as traffic spike, may be lost. Increasing polling frequency may reduce such gap, however, it increases the load on switch CPU significantly. This might also violate the SLAs imposed by tenants on the network service providers (NSPs). Hence, it is desired to have a mechanism which operates in the data plane and can collect fine-grained information at line rate, without significant overheads.

Many solutions have been proposed for network monitoring in academia as well as industry recently that operate at data-plane [25,26]. However, existing solutions mainly focus on trade-off between expressiveness, accuracy, speed and scalability [6,11]. For example, systems such as NetQRE [14] and others can support a wide range of queries using stream processors running on general-purpose CPUs, but they incur substantial bandwidth and processing costs to do so. Telemetry systems such Chimera [13] and Gigascope [27] are expressive in nature by covering wide range of telemetry items, however, can only support lower packet rates. This is because these systems process all packets at the stream processor which can become a bottleneck.

On contrary, telemetry systems that rely on programmable switches alone can scale to high traffic rates [28,29]. However, they can accommodate a limited set of telemetry items in order to achieve the scalability. For example, OpenSketch [30], Sketchvisor [31] and UnivMon [32] can perform telemetry tasks by executing queries solely in the data-plane at line rate, but the queries that they can support are limited by the computational capabilities and memory in the data-plane, scarifying the expressiveness and accuracy. Systems such as ElasticSketch [33], Marple [34] obtain a good balance between the expressiveness and scalability, however, they incur substantial processing overheads, delays and traffic overheads. To overcome this problem, in this work, we propose an approach to minimize the overhead associated with monitoring so as to make the underlying monitoring framework scalable as well as expressiveness.

With the recent advances in programmable data-planes such as P4 [27] along with proper compiler and hardware support [35,36], collecting fine-grained telemetry items from the data-plane is possible at line-rate using e.g. In-band Network Telemetry (INT) [37,38]. While telemetry systems that rely on programmable switches alone can scale to high traffic rates, they give up expressiveness to achieve this scalability. Yu et al. [39] as well as Lahmadi and Boeglin [40] propose use of piggyback technique to monitor network statistics. However, we argue that recent traffic patterns in network slices, with emergence of NFV and SFC architecture, are becoming unpredictable and hence might come in bursts [5]. Due to this, piggybacking may fail to deliver accurate per-flow statistics at the required fine-grained intervals as per the SLAs. Hence, in this work, we advocate active telemetry probing [15], since it is an effective way to perform network monitoring. It is especially effective in the dynamic network slicing architecture,

since each service chain may be allocated different network slices with different QoS requirements undergoing different treatment in the data-plane [28]. We argue that inserting separate network monitoring probes allows the flexibility to collect the network information at fine-grained traffic intervals along the fixed path allocated to the network slice [41]. It also allows to deploy the monitoring flows in more customized as well as optimized manner enabling us to achieve reduced monitoring overheads, as discussed in Section 4 later in this work.

Inserting active probes, however, can be expensive and may lead to queue buildups, buffer-bloat, packet drops and network congestion as well as delays, especially if it is performed in unplanned ad-hoc manner. To minimize the overhead associated with active probing, in this work, we propose mixed integer linear programming (MILP) model for our IntOpt framework. Since MILP cannot solve the larger problem instances without significant computational overheads, we develop a simulated annealing based random greedy meta-heuristic (SARG) approach, that determines the set of monitoring flows (MFs) in order to fulfill all the monitoring requirements of service flows (SFs) in close-to-optimal manner. In the subsequent sections we discuss our proposed IntOpt architecture along with the proposed SARG meta-heuristic in more details.

3. IntOpt architecture and active probing

3.1. INT — Inband Network Telemetry

In this sub-section, we describe the proposed IntOpt architecture. The proposed IntOpt architecture is shown in Fig. 1. IntOpt controller communicates with SDN controller using East–West interfaces. It retrieves information such as underlying physical topology, physical links as well as the service flow requests to be deployed, their actual deployment over the physical nodes and their monitoring demands from the SDN controller (black double-dashed line from SDN controller). The figure shows a network topology with six (SW1 to SW6) physical switches. Also, two service flows, flow A–Z and flow B–Y are deployed through these switches as shown in Fig. 1. Please note we use the terms service function chain (SFC) and service flow (SF) interchangeably.

The IntOpt controller maps service flow (SF) telemetry items and frequency demands to the respective physical links. It then finds out the optimal probing frequency as well as total telemetry items which need to be monitored for each physical link in order to cover all SFs with minimal overhead at the data plane as well at the controller. The controller then prepares an optimal set of monitoring flows (MFs) so that all the given SFs are monitored along with their monitoring demands in terms of telemetry frequency and telemetry items (as explained in the next sub-section). The IntOpt controller performs these tasks by executing the SARG meta-heuristic proposed in this work. The details of the SARG meta-heuristic are explained in Section 5.

The IntOpt controller then identifies telemetry sources (SW1 in Fig. 1), forwarders (SW2–SW5) as well as sinks (SW6), and commands the SDN controller to populate flow tables accordingly (black dashed line). Since we are using active telemetry probing to monitor the network, IntOpt also commands SDN controller to send the periodic monitoring probes for each monitoring flow as per the telemetry frequency determined in the previous step (black double-dashed line to SDN controller), to meet the telemetry demand frequency. P4 programmable switches are responsible for parsing the monitoring probes, inserting the telemetry items and forwarding of the probes to correct output port (red dotted line). The Controller acts as data-sink by instructing the sink switch (SW6) of each monitoring flow to forward the collected information to itself through SDN controller (red double dashed line). It then maps the collected telemetry information back to individual SF requirements to check any SLA violation, such as exceeding the total delays or buffer queue size at the intermediate switch etc. INT controller is responsible for executing the proposed algorithms fast enough to meet real-time requirements of the SFCs. For more details on transmission, storage and processing of decapsulated telemetry information at the INT controller as well as the INT controller capabilities, please refer to our work [30]. When a link or device fails (such as the switch itself), IntOpt controller notices missing monitoring probes due to discontinuous probe IDs. This prompts the controller to inform IntOPt framework about broken service and the associated links (please refer to Section 5-A for more details). Based on the configuration options, IntOpt framework may prompt the administrator to intervene or automatically transfer the service flow and the monitoring flow on the recovery path.

3.2. Optimized active telemetry probes generation

In this sub-section, we illustrate our concept of preparing optimal monitoring flows (MFs) with a toy example. We have considered six different service function chains (SFCs) as service flows (SFs) as shown in Fig. 2, with 15 virtual functions (VFs), numbered from 1 to 15. The numbers on SFC blocks (inside circles and rectangles) indicate the VFs that particular SFC is comprised of. Please note that SFCs may share the VFs. SFCs may have different sizes and shapes as shown in Fig. 2. This may be due to back and forth traffic flows among the VFs.

While deploying the monitoring flows, however, we only consider linear flows. That is, we do not allow forking, or loop formation in MFs. The probes can simply be forwarded on to the port which is inserted as a next hop in telemetry header. Since we propose to perform mapping and extraction of telemetry data at the controller, similar to the approach proposed in [11], implementing linear MFs reduces the controller overhead while preparing the MFs as well as gathering the collected information from the probes and mapping it back to the SFs. Also, this is typically handy allowing the probe forwarding logic at the switches to be simple and fast.

Let us consider a 15-node Atlanta topology from SNDLib [42] for deployment of the six SFs. A possible deployment of SFs over the given substrate network is given in Fig. 2. To illustrate our approach, we focus on a specific SF, SF1, with blue rectangular VFs. Deployment of SF1 over the substrate network is shown in Fig. 2 with double dashed blue lines. Let us assume for simplicity, we implement separate monitoring flow for each SF, following its exact shape on the substrate network. As a result, monitoring flow MF1 for SF1 will also follow the same path as SF1. As we notice, at node 15, the MF1 has to split and the probes gets forwarded to two nodes, node 8 and 9. If we aim to implement simple "next-hop look-up and forward" functionality at the intermediate switches, then it becomes complex to keep track of the next forwarding port due to split-up of the MF. One way to achieve this is to let the controller keep track of switch unique ID (such as MAC) and its forwarding port on that switch for that MF and embed the whole information in the probe. Each switch then performs the match and forwards the monitoring probe accordingly.

The major drawback with such scheme is the delays incurred due to the processing overhead at intermediate switches. Also, the probe size increases as more data needs to be embedded (here MAC of the switch) at every hop, adding to the overhead. Alternatively, we can implement two linear MFs, that is, MF1 with path as (15-9-10-7-14) and MF2 with path (15-8-1-7-14). In this case, forwarding is linear and simple. A simple next-hop port number can be inserted in the probe to guide the intermediate switch to forward the probe to the next hop. However the solution is still non-optimal as the link $E_{7,14}$ is covered twice unnecessarily by the probes. Such overhead due to non-optimal deployment of monitoring flows may increase significantly with increase in the number of service flows and complex physical topology. In this simple case with a single SF, the optimal solution would be to deploy two linear MFs with paths as (15-9-10-7-14) and (15-8-1-7). In our example, we need minimum five MFs for the given set of SFs, as shown in Fig. 3. As we note that, all the used physical links in the topology of Fig. 2 are covered by at least one MF in Fig. 3. Please note that mapping between SFs and MFs in not necessarily 1-to

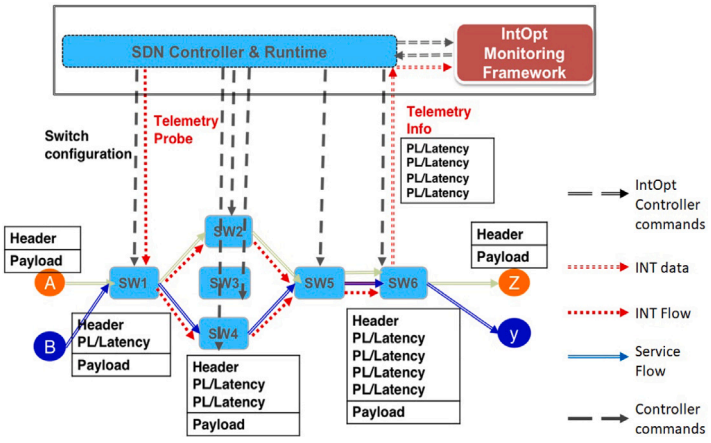


Fig. 1. IntOpt Architecture.

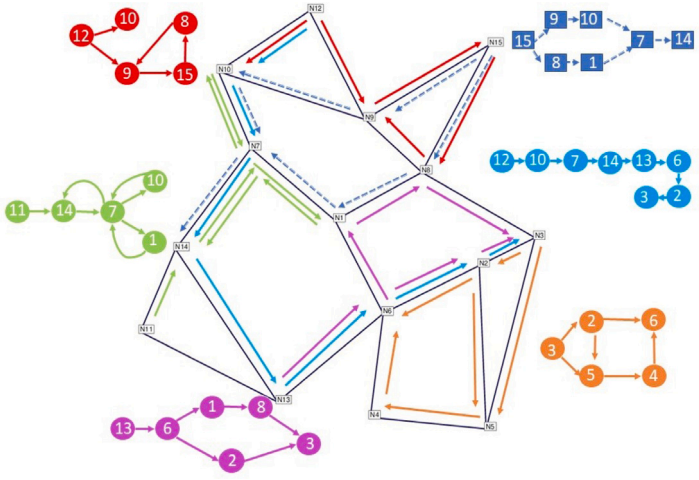


Fig. 2. Service flows and deployment over Atlanta network. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

–1. That is, one monitoring flow may cover more than one SFs or one single SF may be split into more than one MF. This is to minimize the MF overheads and to accommodate as many physical links as possible in one single MF. Please refer to the model presented in Section 4 (especially constraint 4 and 5). This is the reason, though node 9 is the sink of the blue MF, although it is not part of the SF as shown in Fig. 2 (green). We cast this as an optimization problem and develop a simulated annealing based random greedy meta-heuristic (SARG) approach to prepare optimal set of MFs as explained in the next section.

3.3. P4FPGA bench-marking

A VNF generally has incoming and outgoing network interfaces to communicate with other VNFs, that is, to send/receive data packets, in order to complete the service, or in other words, to satisfy the user demand. We argue that it is important to evaluate the performance of the single VNF pipeline, as it would enable researchers in the industry as well as academia to benchmark the monitoring overheads for a single VNF with P4 [1–3]. Also, it is equally important for network researchers and administrators to be aware of the core as well as the edge delays separately, as it allows them to design the edge and core part of the network separately [3,30]. This section presents a bench-marking setup representing a generic VNF, specifically its network interfaces. We also demonstrate that a linear fit to bench-marking results will allow the researchers to extrapolate the monitoring overheads for complex and larger SFC topologies, even with limited knowledge of the SFCs beforehand.

INT capable switch performs following tasks while monitoring probes are passed through them, which cause the significant delays are (1) encap/decap of monitoring probes with telemetry headers at the source and the sink and (2) parse telemetry header and insert telemetry items accordingly (which we call Forwarding or FW for simplicity). In order to approximate the processing delays induced by INT-operations, we performed a series of experiments using the NetFPGA-SUME hardware platform. In this sub-section, we demonstrate our setup and results to bench-mark the total monitoring overheads. Specifically, we aim to benchmark a co-relation between total monitoring overheads for all the monitoring flows and actual delays incurred due to such overheads by bench-marking the P4 implementation with P4 NetFPGA. We then use these bench-marking results to compare our proposed SARG scheme with ad-hoc naïve approach and present our detailed results for larger problem instances later in this section.

We have implemented a P4 program using the P4FPGA toolkit [17], based on the INT specification, which parses incoming packets, and pushes INT headers (encapsulation), accordingly [37]. The headers are divided into two types, (1) Telemetry Instruction and (2) Telemetry Data headers. Instruction headers contain a set of instructions that determine, which telemetry data items should be pushed by each switch, along with various meta-data. Telemetry data headers contain actual INT data such as queue occupation, switch traversal latency, etc. The P4 INT program checks incoming packets for instruction headers, and

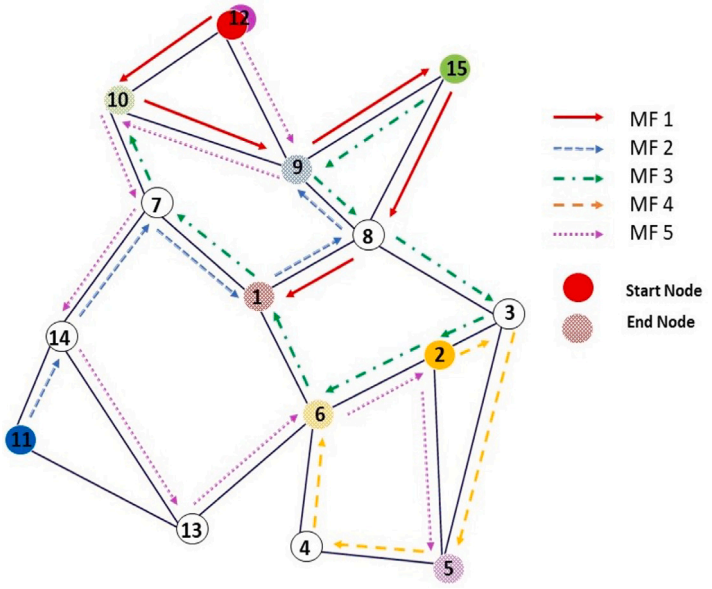


Fig. 3. 5 monitoring flows to cover all the service flows.

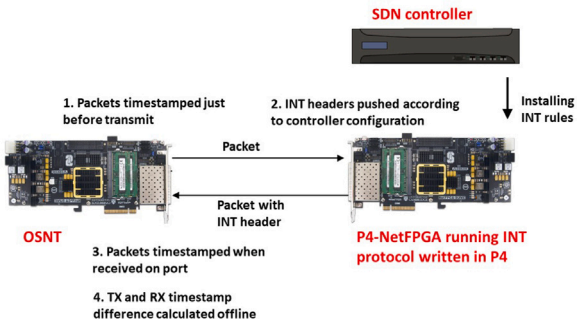


Fig. 4. NetFPGA setup for Core and edge switches.

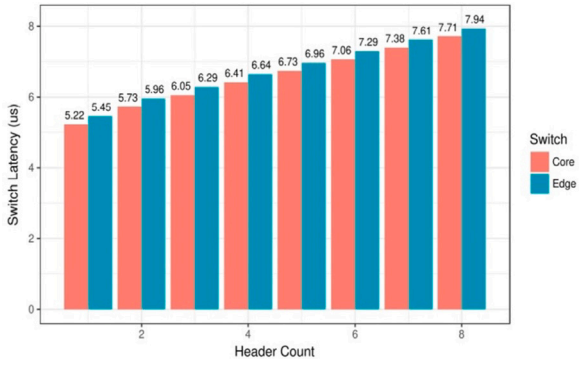


Fig. 5. NetFPGA results for Core and edge switches.

if found one, it pushes the telemetry data specified in the instruction header. Should a packet arrive without an instruction header, the switch can be configured through the control plane, to either forward it as normal packet or insert a telemetry instruction header (source node). This is configured through a match key, such as source port or destination address.

We ran experiments with both edge and core switch P4 programs, pushing from zero to eight INT data headers. Edge switches were configured to push both the instruction header and the configured number of data headers, while the core switches only pushed data headers. The setup is shown in Fig. 4. We use OSNT as a traffic generator due to its nanosecond granularity. OSNT timestamps packets while sending and receiving them which allows us to calculate the desired delays incurred due to INT header operations. Fig. 5 shows the experienced delays against the number of header fields inserted for core as well as edge switches. As we observe, there is a linear relationship between the telemetry data pushed in the packet and the delays observed. The edge switch has higher latency due to additional push operation for instruction headers. This can be justified with the fact that edge switches are responsible for more complex configuration and security tasks, such as MPLS/BGP handling etc. Due to such complex operations, INT headers experience more delays at edge switches.

3.4. ONAP-IntOpt integration

Integrating InOpt into a network management platform such as Open Networking Automation Platform (ONAP) [43], can be done in two ways described in this subsection. The first option is to implement IntOpt on the SDN controller level, the approach advocated in this work (Fig. 1). The advantage of this approach is that IntOpt framework remains more generic and can easily be integrated with any cloud management platform or the framework. However, separate APIs need to be developed for successful communication among components of the cloud management platform and IntOpt modules. Here, ONAP would decompose an overall service request into one or several function chains and include the monitoring requirements in the flow setup. All the optimization and analysis would be done at the SDN controller level. The resulting monitoring data and messages will be consumed by ONAP Data Collection, Analytics, and Events (DCAE) module, using already defined protocols.

For the second option, which is more ONAP specific, we propose utilizing ONAP components to realize different functions of the InOpt framework. This would enable us to create monitoring flows across multiple SDN controllers (SDNCs) and take actions on a network-wide scale. The InOpt modules would be implemented within the DCAE framework to keep track of monitoring requirements, optimization of the probe locations and analysis of the data from the probes. It means, the requirement of developing the communication APIs between IntOpt and the components of ONAP would be minimal. This also allows us to reuse the existing functionality of the ONAP components for the common functionality, such as database management and others.

In addition, there are functionality within ONAP to create automated control loops, where policy actions can be triggered based on measured performance. Performance data from InOpt could be used as input to such control loops, along with other types of measurements and events from the network.

4. Optimization model

In this section, we formulate an optimization model using mixed integer linear programming (MILP) to minimize the total monitoring overhead associated with the proposed active probing monitoring scheme using INT framework. The symbols used in the formulation are explained in Table 1.

4.1. Objective function

Optimization function is given in Eq. (1). Please note that, though the two items in the optimization functions may utilize different hardware resources, in Section 6, we demonstrate that, their overheads are comparable and thus the sum can be mapped to hardware resource utilization [44].

Minimize:

$$\sum_{f=1}^{F}(a_f \times \alpha) + \sum_{j=1}^{N}\sum_{i=1}^{N}\sum_{f=1}^{F}(\phi_{ij}^{f} \times a_f \times \delta_{ij} \times \beta) \tag{1}$$

First term in Eq. (1) represents overheads associated with preparing the monitoring flows, that is encap–decap overheads (encapsulation/decapsulation of the MFs at the source and the sink). The second terms represents the overheads associated with header lookup as well as insertion of the telemetry data in the monitoring probes, in terms of delays, along the path of all MFs. ϕ_{ij}^{f} represents the links used for MF f and δ_{ij} represented the monitoring demand of the link represented by ij.

4.2. Constraints

We aim to achieve the objective in Eq. (1), given all the below constraints are satisfied. Input variables δ and μ are explained in more depth in Section 5.

(1) *Flow conservation constraint:*

- *Source Node:* Source node of a MF f must not receive any data/packet from any other node in the given MF f.

$$\sum_{j=1}^{N}\phi_{ji}^{f} = 0, \forall_f \in F, \forall_i \in \Psi \tag{2}$$

- *Destination Node:* Destination node of a MF f should not forward the data/packet received to any other node in the MF f.

$$\sum_{j=1}^{N}\phi_{ij}^{f} = 0, \forall_f \in F, \forall_i \in \Delta \tag{3}$$

- *Intermediate Node:* Any Intermediate node must forward the MF data/packet if and only if it has received any.

$$\sum_{j=1}^{N}\phi_{ij}^{f} = \sum_{j=1}^{N}\phi_{ji}^{f} \tag{4}$$

$$\forall_f \in F, \forall_i \in N, \forall_i \notin \Delta, \Psi$$

(2) *Link Integrity constraint:* Any monitoring flow or a service flow should be passed between two nodes i and j if and only if there is a physical link between i and j.

$$\phi_{ij}^{f} \leq \xi_{ij}, \forall_f \in F, \forall_{ij} \in N \tag{5}$$

$$r_{ij}^{s} \leq \xi_{ij}, \forall_s \in S, \forall_{ij} \in N \tag{6}$$

Table 1
Optimization model symbols.

Symbol	Description
Indices	
i, j, k, l	Iterators for nodes in the topology s.t. $i, j, k, l \in N$
f, g	Iterators for monitoring flows in the topology s.t. $f \in F$
s, t	Iterators for service flows in the topology s.t. $f \in F$
Input Constants	
N	Total number of Nodes in the physical topology
F	Maximum number of total monitoring-flows that may be deployed in the system.
S	Total service flows routed in the network
α	Active monitoring probe encap–decap overhead in terms of delays associated.
β	Overhead associated with the instruction header lookup as well as insertion of the telemetry data in the monitoring probe, in terms of delays associated.
Input vectors	
Δ	Telemetry item demand 2-D matrix for physical links s.t. $\Delta = \{\delta_{ij} \in \tilde{N}\}$ and $\lvert\Delta\rvert = N \times N$
$\tilde{\mu}$	Telemetry frequency demand 2-D matrix for physical links s.t. $\tilde{\mu} = \{\mu_{ij} \in \tilde{N}\}$ and $\lvert\tilde{\mu}\rvert = N \times N$
Ξ	Physical connectivity 2-D matrix s.t. $\Xi = \{\xi_{ij} \in \{0,1\}\}$ and $\lvert\Xi\rvert = N \times N$ $\xi_{ij} = \begin{cases} 1, & \chi_1 \\ 0, & \text{otherwise} \end{cases}$ χ_1: if there is physical link between node i and node j.
R	Service flow routing 3-D binary matrix, s.t. $R = \{r_{ij}^{f} \in \{0,1\}\}$ and $\lvert R\rvert = S \times N \times N$ $r_{ij}^{f} = \begin{cases} 1, & \chi_2 \\ 0, & \text{otherwise} \end{cases}$ χ_2: if node i forwards SF f to node j.
Λ	MF threshold 1-D integer vector, s.t. $\Lambda = \{\lambda^{f} \in \tilde{N}\}$ and $\lvert\Lambda\rvert = F$
Γ	Physical link threshold 2-D integer vector, s.t. $\Gamma = \{\gamma_f \in \tilde{N}\}$ and $\lvert\Gamma\rvert = N \times N$
Decision vectors	
Φ	Monitoring flow routing 3-D binary matrix, s.t. $\Phi = \{\phi_{ij}^{f} \in \{0,1\}\}$ and $\lvert\Phi\rvert = F \times N \times N$ $\phi_{ij}^{f} = \begin{cases} 1, & \chi_3 \\ 0, & \text{otherwise} \end{cases}$ χ_3: if node i forwards MF f to node j.
$\tilde{A}$	MF activation 1-D binary vector, s.t. $\tilde{A} = \{a_f \in \{0,1\}\}$ and $\lvert\tilde{A}\rvert = F$ $a_f = \begin{cases} 1, & \chi_4 \\ 0, & \text{otherwise} \end{cases}$ χ_4: if MF f is activated for monitoring.
Ω	Link activation 2-D binary matrix, s.t. $\Omega = \{\omega_{ij} \in \{0,1\}\}$ and $\lvert\Omega\rvert = N$ $\omega_{ij} = \begin{cases} 1, & \chi_5 \\ 0, & \text{otherwise} \end{cases}$ χ_5: if link e_{ij} is used for some MF.
Θ	2-D integer matrix to check the loops, s.t. $\Theta = \{\theta_{ij} \in N\}$ and $\lvert\Theta\rvert = N \times N$
Ψ	A 2-D binary matrix indicating source node for each data-flow $\Psi = \{\psi_{i}^{f} \in \{0,1\}\}$ and $\lvert\Psi\rvert = N \times F$ $\psi_{i}^{f} = \begin{cases} 1, & \chi_6 \\ 0, & \text{otherwise} \end{cases}$ χ_6: if node i is a source node for MF f.
$\rho_1, \rho_2, \rho_3, \rho_4$	Binary variables to hold intermediate variable values, s.t. $\rho_1, \rho_2, \rho_3, \rho_4 = \{0, 1\}$

(3) *MF convergence and non-forking constraint:* A single monitoring flow cannot be forwarded to two different nodes from a single node, that is no forking of any MF should be allowed. Similarly, no node can receive a single MF from two different nodes.

$$\sum_{j=1}^{N} \phi_{ij}^{f} \leq 1, \forall_f \in F, \forall_i \in N \tag{7}$$

$$\sum_{i=1}^{N} \phi_{ij}^{f} \leq 1, \forall_f \in F, \forall_j \in N \tag{8}$$

(4) *Link activation constraint:* We call a physical link ξ_{ij} activated if and only if it carries the data of any monitoring flow f. We prepare a binary matrix Ω indicating whether the link is activated or not. For example, Eq. (9) makes sure that if ϕ_{ij} is equal to one for any flow f, then ω_{ij} should be one. Also, vice-a-versa, Eq. (10) checks if ω_{ij} is one, that is, if the link ξ_{ij} is activated, then ϕ_{ij} should be equal to one for all flows passing through that link. This condition will be used later to formulate some other constraints.

$$\sum_{f=1}^{F} \phi_{ij}^{f} \geq \omega_{ij}, \forall_{ij} \in N \tag{9}$$

$$\phi_{ij}^{f} \leq \omega_{ij} \tag{10}$$

$$\forall_f \in F, \forall_{ij} \in N$$

(5) *Monitoring flow activation constraint:* We maintain a separate variable $\tilde{A}_f$ to indicate activation of monitoring flows, which is used in the optimization function. Below constraint makes sure that appropriate flows are activated as per the monitoring flow routing matrix.

$$a_f \geq \phi_{ij}^{f}, \forall_f \in F, \forall_{ij} \in N \tag{11}$$

$$a_f \leq \sum_{i=1}^{N} \sum_{j=1}^{N} \phi_{ij}^{f}, \forall_f \in F \tag{12}$$

(6) *Monitoring flow capacity constraint:* At each hop, telemetry items are inserted in a monitoring flow as per the monitoring demand of the link. Total number of telemetry items inserted in a monitoring flow should be less than a pre-defined threshold to avoid the MF becoming bulky. A bulky MF may affect the network performance adversely. We take the threshold value as an input to the model. The constraint is modeled as follows.

$$\sum_{i=1}^{N} \sum_{j=1}^{N} (\phi_{ij}^{f} \times \delta_{ij}) \leq \lambda^{f}, \forall_f \in F \tag{13}$$

(7) *Physical link capacity constraint:* To avoid a link congestion due to the monitoring traffic we limit the total number of MFs passing through one single link. We take the threshold value for each link as an input vector of size $N \times N$. The constraint is modeled as follows.

$$\sum_{f=1}^{F} \phi_{ij}^{f} \leq \gamma_{ij}, \forall_{i,j} \in N \tag{14}$$

(8) *Service flow unity constraint:* This constraint makes sure that each physical link used for every service flow is covered by at-least one MF, that is, value of ω_{ij} for each link ξ_{ij} should be greater than SF routing value for link ξ_{ij}. Combined with link activation constraint, this eventually makes sure that all SFs are monitored by at-least one MF.

$$\omega_{ij} \geq R_{ij}^{s} \tag{15}$$

$$\forall_f \in F, \forall_{i,j} \in N$$

(9) *No-loop constraint:* It is a known fact in graph theory that a digraph $G = (V, E)$ is acyclic if and only if its vertices can be assigned numbers from 1 to $|V|$ in such a way that $\theta[i]+1 <= \theta[j]$ for every arc (i, j) in E, where $\theta[i]$ is a number assigned to vertex i. The constraint is modeled as shown in the equation below. It makes sure that if there is a routing from node i to node j for flow f, then value of θ_j^f should be greater that θ_i^f. This will make sure that there are no-loops, given value of Θ cannot be greater than N.

$$(1 - \phi_{ij}^{f}) + (\theta_j^f - \theta_i^f) \geq 1, \tag{16}$$

$$\forall_f \in F, \forall_{i,j} \in N$$

(10) *MF Frequency integrity constraint:* This constraint makes sure that the links covered by a specific MF have consistent monitoring frequency demands for the service flows routed over them. We achieve this by forcing the frequency demands of the links other than the first link of the MF to be less than equal to the frequency demand of the first link. The more strict constraint would be having frequency demand for all links in a given MF to be equal. The constraint is modeled using a set of equations below.

$$\psi_k^f \leq M_1 \times (1 - \rho_1) \tag{17}$$

$$\phi_{kl}^{f} \leq M_2 \times (1 - \rho_2) \tag{18}$$

$$\phi_{ij}^{f} \leq M_3 \times (1 - \rho_3) \tag{19}$$

$$\mu_{kl} - \mu_{ij} \leq M_4 \times (\rho_1 + \rho_2 + \rho_3) \tag{20}$$

$$\forall_f \in F, \forall_{i,j,k,l} \in N$$

It is to be noted that M_1, M_2, M_3, M_4 are large integer constants.

4.3. MILP complexity analysis

We now analyze the complexity of the proposed MILP model. It is easy to notice that the complexity of the proposed model is bounded by the processing complexity of the 3-D decision variable Φ. All the constraints, except the constraint 10, *frequency integrity constraint*, iterate through variables F and twice through N. However, as seen from Eqs. (17)–(20) of constraint 10, we observe that, the iteration through N happens four times, using iterators i, j, k and l. Hence, the total complexity of the MILP turns out to be $O(N^4)$. Due to such high computational complexity, we could obtain the results using MILP for a smaller physical topology with lesser number of service flows, as explained in Section 6. To solve larger instances of real-time problems within acceptable time frame, we propose simulated annealing based random greedy (SARG) approach in the next section.

5. Algorithms for active probing optimization

In our previous work, we have demonstrated that the application of the general theory of optimization by random search gives us near-optimal results [45]. The mathematical treatment of this technique is given in [46]. The random search algorithm implemented in SARG approach belongs to the category of Global Optimization. In contrast with the deterministic methods like branch and bound which guarantee asymptotic convergence to the optimum at the high computational effort, random search and population based meta-heuristics find a relatively good solution quickly [47] and are usually easy to construct. Hence, in this section we develop simulated annealing based random greedy meta-heuristic that determines the optimal set of monitoring flows (MFs) in order to fulfill all the monitoring requirements of service flows (SFs) while minimizing the overhead. We also benchmark the INT framework using P4FPGA, as explained in Section 3-C.

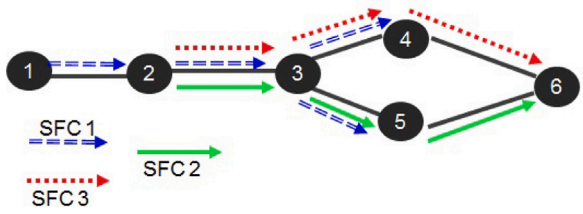


Fig. 6. Service Flow demands to link demands mapping.

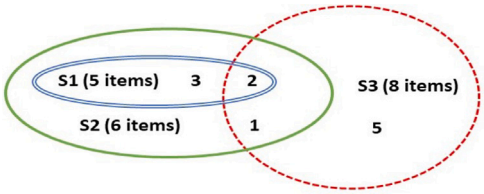


Fig. 7. Telemetry item demands for service flows.

Table 2
Service flow frequency and telemetry demands.

Service flow	Frequency (ms)	Telemetry items
SFC1	5	5
SFC2	1	6
SFC3	10	8

5.1. SARG approach

In this sub-section, we explain our SARG approach to prepare near-optimal set of MFs. As a first step towards preparing the optimal set of MFs, we prepare set $\tilde{E}$ of physical links to which logical links of service flows are mapped. Then we map the SF telemetry items and frequency demands to the respective physical links in $\tilde{E}$ and design minimum number of monitoring flows (MFs) to monitor all those links. To achieve this, the heuristic prepares two sets, μ_{ij} and δ_{ij}. μ_{ij} denotes a strict bound for the telemetry frequency demands and δ_{ij} denotes a strict bound for the telemetry item demands for all SFs passing through the link E_{ij} ($E_{ij} \in \tilde{E}$). We implement a pre-processing step at the controller and maintain separate data structures to keep track of μ_{ij} and δ_{ij}.

We now demonstrate the aforementioned steps of the SARG algorithm with a simple substrate network and three service flows (SFs) as shown in Fig. 6. Let us denote the sets of telemetry items demanded by SFC1, SFC2 and SFC3 as set S_1, S_2 and S_3 respectively. The frequency and telemetry item demands for each SF are given in Table 2. For example, SFC1 demands 5 telemetry items and the frequency of the telemetry is desired to be 5 ms. Similarly it holds for other two SFCs as well. *Venn* diagram shown in Fig. 7 demonstrates telemetry item demands for each SFs. As shown in the figure, S_2, a set of 6 telemetry items demanded by SFC2 is a super-set of S_1, 5 items demanded by SFC1. However, S_3 for SFC3 is intersecting S_1 and S_2 as shown in Fig. 7, with 2 items in common with SF1 and 1 more item in common with SFC2 (in total 3 items common with SFC2). For example, S_1={switch ID, ingress and egress timestamps, throughput, queue buffer sizes, path followed by packets (port numbers)} and S_2={switch ID, ingress and egress timestamps, throughput, queue buffer sizes, path followed by packets (port numbers), queue size}. In this example all the telemetry items in S_1 are covered by S_2. Hence, MF covering S_2 will also be covering the demands of S_1.

The pre-processing step maps SF monitoring demands to link demands and fills sets μ_{ij} and δ_{ij} as shown in Table 3. As we observe, if only one service flow is passing through the link, then telemetry frequency demand mappings are straightforward (such as links E_{12}, E_{56} and E_{46}). However, for links accommodating more than one SF, we need to determine the appropriate mappings. For example, monitoring frequency demand for link E_{23} should be 1 ms since, from Table 2, we observe that it is the most strict telemetry frequency demand for all the SFs passing through link E_{23}. Also, it will cover the telemetry frequency demands for other SFs, which are greater than 1 ms. Similarly, for link E_{34}, the monitoring frequency demand should be 5 ms as it is more strict (5 ms) compared to the other (10 ms), and so on.

Table 3
Frequency demands mappings.

Link	MF Frequency in ms (μ_{ij})	Telemetry Items (δ_{ij})
E_{12}	5	S_1
E_{23}	1	$S_4 = S_1 \cup S_2 \cup S_3$
E_{34}	5	$S_5 = S_1 \cup S_3$
E_{35}	1	$S_6 = S_1 \cup S_2$
E_{46}	10	S_3
E_{56}	1	S_2

Similarly, we map the telemetry item demands of the SFs to the link demands (column 3 in Table 3). For example, we observe that on the link E_{23}, the set S_2 with 6 telemetry items for SF2 also covers S_1 with 5 items for SF1 (as S1 ⊂ S2 as mentioned earlier). However, every 1 ms, we need to insert a new set S_4 of telemetry items in the monitoring flow over E_{23} such that $S_4 = S_1 \cup S_2 \cup S_3$. This is because SF3 has a few telemetry items in S_3 which are not covered by S_2. As we can see, the size of S_4 is 11 telemetry items (since three items are common between S_2 and S_3).

Once the mapping stage is completed, *Random Greedy* procedure maps link monitoring demands to MFs so that all SFs with the specified SLAs are covered while the number of total MFs are minimized. Algorithm 1 illustrates the steps for our Random Greedy policy. The heuristic begins by initializing an empty monitoring flow and adding any random link from $\tilde{E}$ to it. We call two links as *neighboring links* if they share one node in common. We keep adding more links to the MF by selecting neighboring links sequentially from set $\tilde{E}$ thereafter, given the link being added has similar or less strict monitoring demands than the existing set of links in the given MF. Selected link is then removed from $\tilde{E}$. This is repeated until the size of the MF grows beyond the threshold. At that instance, the heuristic terminates the monitoring flow and start a new one. The process is repeated until all links in $\tilde{E}$ are covered, that is, set $\tilde{E}$ becomes empty.

We also propose a simulated annealing based meta-heuristic approach as shown in Algorithm 2, which prevents the random greedy approach from getting stuck in local-minima. The *quality* attribute of the solution returns the total number of encap/decap plus forwarding instances at the data plane due to the proposed MF deployment scheme in the given solution. It has been used as the fitness function for comparison of the solutions.

We have implemented an ad-hoc approach as well, which we call as *naïve* algorithm, which is unaware of the optimization policies and widely used in the current practical use. *naïve* algorithm just tries to avoid forking or looping of the MFs, which is the basic requirement for MF to be a valid flow. In the naïve implementation, we just start the MF for each SF and follow it linearly. If there is any forking or loop formation in the SF, we just break the existing MF and form a new one. This is the typical approach which is generally followed in the absence of any sophisticated algorithm for the MFs formation. Steps for the naïve approach are given in Algorithm 3.

5.2. Heuristic complexity analysis

As explained in the beginning of this section, deterministic methods for global optimization are NP-hard, however, on contrary, a random search method may be executed in polynomial time. However, due to large number of physical nodes and links associated with physical networks, even the polynomial time complexity results in unacceptable

Algorithm 1 Random Greedy Approach integrated with Simulated Annealing

procedure RANDOM_GREEDY
  $\tilde{E} \Rightarrow$ set of the edges covered by all service flows
  $\lambda_f \Rightarrow$ monitoring flow size threshold
  $E_{ij} \leftarrow Random\_Select(\tilde{E})$
  Initialize a monitoring flow m_f
  Monitoring Frequency(m_f) $\leftarrow \mu_{ij}$
  Telemetry Items(m_f) $\leftarrow \delta_{ij}$
  Set E_{ij} as a start link of m_f
  while $\tilde{E} \neq \Phi$ **do**
    $E_{jk} \leftarrow Sequential\_Select(\tilde{E})$
    if $\mu_{ij} \leq \mu_{jk}$ and $\delta_{ij} \geq \delta_{jk}$ **then**
      $m_f \leftarrow m_f + E_{jk}$
      $\tilde{E} \leftarrow \tilde{E}$ - E_{jk}
    else
      terminate m_f and initiate new flow m_{f+1}
      break the for loop
    end if
    if $Size(m_f) > \lambda_f$ **then**
      terminate m_f and initiate new flow
      break the for loop
    end if
  end while
end procedure

Algorithm 2 Simulated Annealing meta-heuristic steps

procedure SIMULATED ANNEALING
  $temperature \leftarrow \lambda$
  $cooling\_rate \leftarrow \alpha$
  $previous\_sol \leftarrow new\_sol \leftarrow best\_sol \leftarrow NULL$
  while $temperature > 1$ **do**
    $new\_sol \leftarrow Random\_Greedy\_Procedure()$
    if $e^{(new\_sol.quality - prev\_sol.quality)/\lambda} > Random(0,1)$ **then**
      $prev\_sol \leftarrow new\_sol$
      if $new\_sol.quality > best\_sol.quality$ **then**
        $best\_sol \leftarrow new\_sol$
      end if
    end if
    $temperature \leftarrow temperature \times (1 - cooling\_rate)$
  end while
end procedure

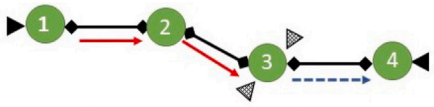
delays for real-time problems. Hence, for the problem of generating monitoring flows to cover the deployed service flows, we have taken an approach to reduce the complexity of the heuristic by iterating only through the physical links which are actually used to deploy the service flows (that is the links from set $\tilde{E}$) and eliminating the unused physical links from the processing completely. Generally a set of physical links used for the given set of service flows is much smaller than the whole set of physical links, especially in the larger topology, reducing the time-complexity of the heuristic significantly. However, while doing so, we have to make sure that the solution quality is not compromised.

With the two lemma below, we prove that by eliminating unused physical links from preparing monitoring flows, in fact, does not hamper the solution quality, while reducing the execution time significantly. As explained earlier in this section, we select a physical link randomly as a starting point of the MF and keep adding neighboring links to it from set $\tilde{E}$. Considering this we state our first lemma as:

Lemma 1. *To achieve minimum monitoring overhead, MF should be extended to the full extent along the neighboring links, given constraints 6, 10 (Section 4) are satisfied*

Algorithm 3 Naïve ad-hoc Approach

procedure NAÏVE_APPROACH
  $s \leftarrow Sequential\_Select(R)$
  $\tilde{E} \Rightarrow$ set of the edges covered by service flow s
  $\lambda_f \Rightarrow$ monitoring flow size threshold
  Initialize a monitoring flow m_f
  $E_{ij} \leftarrow Sequential\_Select(\tilde{E})$
  Set E_{ij} as a start link of m_f
  while $\tilde{E} \neq \Phi$ **do**
    $E_{jk} \leftarrow Sequential\_Select(\tilde{E})$
    if (no loop or fork) and $\mu_{ij} \leq \mu_{jk}$ and $\delta_{ij} \geq \delta_{jk}$ **then**
      $m_f \leftarrow m_f + E_{jk}$
      $\tilde{E} \leftarrow \tilde{E}$ - E_{jk}
    else
      terminate m_f and initiate new flow m_{f+1}
      break the for loop
    end if
    if $Size(m_f) > \lambda_f$ **then**
      terminate m_f and initiate new flow
      break the for loop
    end if
  end while
end procedure



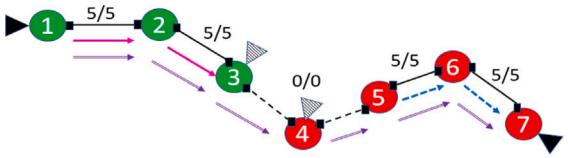
	ENCAP/DECAP	Lookup/Insertion
CASE 1	2	6
CASE 2	4	6

Fig. 8. Lemma 1: Extension along neighboring links.

This can be explained with Fig. 8. As shown in the figure, let us assume that links E_{12} and E_{23} are used by a service flow 1 (SF1) and link E_{34} is used by SF 2, assuming same telemetry and frequency demands for simplicity. Also, let us assume that the monitoring flows have large capacity threshold (for constraint 6), so that we do not have to break the flows. As for case 1, let us assume all three links are covered by the single MF. As shown in the table in the figure, for case one, there will be two Encap/Decap instances (shown by solid triangles) and 6 look-up/Forwarding instances (shown by small squares along the links). Now let us assume we decide to split the MF into two as shown by red-solid and blue-dotted arrows. Using two MFs instead of one incurs additional overhead as shown in case 2 (dot-filled triangles). This confirms the fact that the MF should be extended as much as it can be, along the neighboring links, to have minimum monitoring overhead.

Lemma 2. *Monitoring overhead always increases by combining two non-neighboring links with a non-participant link to extend the MF, even if constraint 6 and 10 are satisfied*

This can be explained with Fig. 9, where there are six links in the topology out of which two links (dotted black) are not used for the deployment of any SF (non-participant links). Let us assume we deploy two MFs to cover these two SFs, indicated by pink-solid and blue-dotted arrows. The monitoring overheads are shown in the table in the same figure. However, let us assume in case 2, we implement a single MF to cover all the used physical links by considering the unused links in the MF and extending the MF as shown in Fig. 9 with double violet line. In this case the monitoring overheads increase as shown in the table in Fig. 9. This confirms the fact that the solution quality is



	ENCAP/DECAP	Lookup/Insertion
CASE 1	4	8
CASE 2	2	12

Fig. 9. Lemma 2: Extension by combining two non-neighboring links with a non-participant link.

not compromised when we consider the links only from set $\tilde{E}$, while reducing the monitoring overhead as compared against considering all the links of the physical topology, from set E.

Based on Lemmas 1 and 2, SARG maps SF link demands to physical links with computational complexity of O(L), where L is the size of the set $\tilde{E}$, actual physical links used for deploying the SFs. SARG then selects a physical link from $\tilde{E}$ and starts a MF. The MF is extended as much as it can be, by adding neighboring links to it from $\tilde{E}$. The complexity of this step is O(L) again. Check for the loop formation can be done in O(1) (constraint 9). MF is broken if frequency or telemetry item demands are mismatched or MF limit is reached (constraints 6 or 10 are violated). This step can also be performed in O(1). Above steps are repeated until all links in $\tilde{E}$ are covered by at least one MF (constraint 8). It means the loop is executed for O(L) times. Hence the total complexity of the random greedy approach becomes O(L^2). If we assume that size of the set $\tilde{E}$ equals N, then the overall complexity of the random greedy stage turns out to be O(N^2). Based on the number of iterations in the simulated annealing stage, the final complexity can be written as O($X \times N^2$), where X is the total number of iterations performed. X is a configurable parameter and the effect of X on the total time-complexity of the proposed meta-heuristic is demonstrated in the next section.

6. Numerical results

In the next section, we analyze the bench-marking results by implementing a P4 program using the P4FPGA toolkit. In addition, we present numerical results obtained through hardware setup that compare the two meta-heuristics with the optimal solution in terms of monitoring overheads and time complexity.

6.1. MILP vs. Heuristic

We first compare the performance of the proposed SARG heuristic with the optimal solution. We have solved the MILP model presented in Section 4 using MiniZinc [48] and CPLEX solver [49]. We choose Atlanta (with 15 nodes and 22 links) as a substrate node topology from SNDLib [42]. Then we choose a set of service flows (SFs) to be deployed along with monitoring requirements such as monitoring frequency as well as telemetry item demands for the given set of service flows (SFs). A set of six random service flows, substrate network topology and deployment of those SFs over the substrate network are shown in Fig. 2. As mentioned earlier, we have considered different possible complex shapes and sizes for the SFs, which are generally observed with different application service providers (ASPs) [50]. The mapping between the virtual links and physical links is random. Please note that, since placement is not the goal of this paper, we do not check for any link capacities as well as delay or cost constraints, however, we just check the feasibility of mappings between physical links and logical/virtual links of the considered SFs. Telemetry frequency demands (in milliseconds) are chosen randomly from a set of {5, 6, 7, 8, 9, 10}.

Table 4
MILP and SARG comparison.

Algorithm	SFs	MF	Encap/-de-cap	FW/-Lookup	Exe. time (s)	Mon. delays (ms)
MILP	5 SFs	4	8	42	448.80	69
	6 SFs	5	10	50	1071.30	75
	7 SFs	8	16	56	8123.00	83
SARG	5 SFs	4	8	44	12.00	72
	6 SFs	6	12	54	20.40	78
	7 SFs	9	18	60	28.50	87

Telemetry item demands (telemetry items per hop) are chosen from a set {5, 6, 7, 8, 9, 10}.

The results are presented in Table 4 in term of total MFs deployed, total lookup and forwarding instances and total encap/de-cap instances, which contribute to the total monitoring overheads. We have also noted down the total execution time taken by the MILP and the SARG heuristic for the execution. SF placement for six sample SFs and its corresponding optimal solution are given in Figs. 2 and 3 respectively. Experiments were performed on Intel Xeon(R) processor with CPU E5-2650, 2.00 GHz having total eight cores and 400 GB of memory. For SARG, we have considered the starting temperature values as 10^5 and cooling rate value as 10^{-5}, which has been observed to be the best, as shown in the detailed results later-on in this section.

As we observe, for the case of (15-node, 22-link topology) and 7 SFCs, MILP deployed 8 MFs to cover all the links, with 16 encap–decap instances, 56 FW/Lookups and total monitoring delays of 83 ms. The proposed heuristic deployed 9 MFs, with 18 encap–decap instances, 60 FW/Lookups and total delays of 87 ms. For all observed cases, performance of the proposed heuristic (in terms of total monitoring delays) is within 5% of the optimal solution, with significantly less execution time. Due to the computational complexity of the MILP model, it takes significant amount of time and hence solution can be obtained for smaller topology only, that is, for Atlanta, and for small number of service flows as well.

6.2. Detailed numerical analysis

We now evaluate the performance of our proposed SARG scheme and the naïve approach, for larger real-time topology and larger number of service flows. We have used the Europe topology from SDNLib which has 37 nodes in total and 178 links [42]. We have generated the service chains randomly, as explained in sub-section A. We also selected specific parameters for SFs such as average length in terms of hops, telemetry items demands and telemetry frequency demands randomly from specific ranges given as an input to the heuristic, explained earlier in this section. We varied the total number of actual SFs to be deployed in the network from 5 to 50, to observe effect on final results. Please note that naïve approach has less processing complexity. However, it may be noted that the solution obtained with the naïve approach is far from the optimal, which may not be acceptable with more stringent SLAs. Also, with the increasing processing power of the INT controllers [30], it is now possible to implement complex algorithms, which can reduce the gap with the optimal solution. Hence we propose more advanced SARG approach to improve the solution quality.

We have considered two different cases for average hop-length and telemetry item demands for the SFs. That is, the first case with average hop-length as 10, telemetry items needed (also called as telemetry lengths) are 5 and telemetry frequency demand as 5 ms. For the second case, the hop-length as 10, telemetry items needed are 10 and the telemetry frequency demand is 10 ms. For simulated-annealing heuristic, initial temperature and cooling rate are kept constant at 10^5 and 10^{-5} respectively, as these values have achieved the best results, shown later in this section. Fig. 10 shows total encap/decap plus forwarding instances (alternatively called as monitoring or telemetry overheads) in

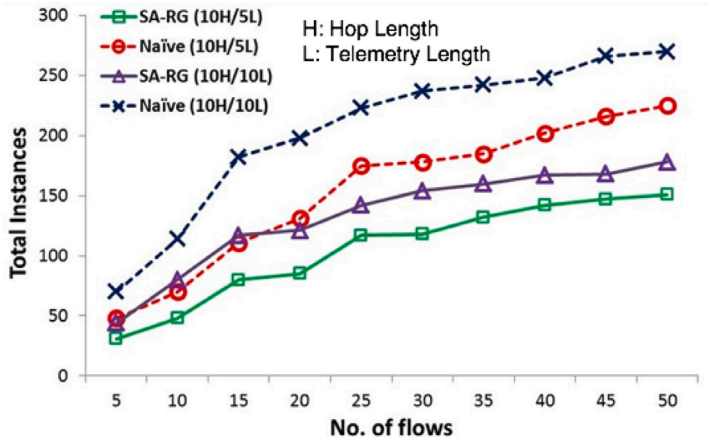


Fig. 10. Total encap–decap and FW instances.

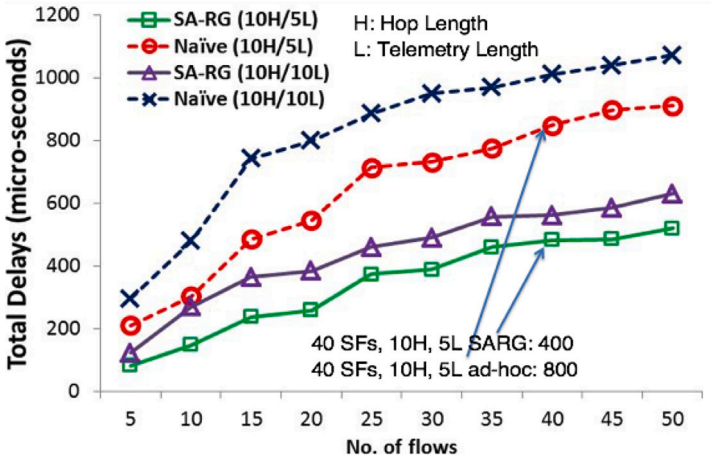


Fig. 11. Total delays against total SFs.

the given network along the Y-axis against different number of service flows along the X-axis. In Fig. 11 we plot the graphs for the total delays incurred due to the processing of monitoring flows. For this purpose, we use the overhead values obtained from the experimental bench-marking of P4 INT framework on the NetFPGA-SUME hardware as discussed in Section 3.3.

As we observe, in Fig. 10, the monitoring overheads are minimum for the proposed SARG scheme. For example, for 25 SFs with case 1 (10H/5L), the number is 100 (green squared solid line), however for the naïve approach the number is 163 (red circled dotted line). For case 2 (10H/10L) the numbers are 122 and 210 for SARG and naïve approach respectively (triangle vs. cross lines). Corresponding delays for case 1 as shown in Fig. 11 are, 300 and 700 micro-seconds for SARG and naïve approach respectively. For case 2 the delays are observed to be 400 and 800 micro-seconds. We also observe that such delays increase with the increase in average hop length of the SFs (green squared solid line vs. purple triangle solid line). In Fig. 12 we keep the total number of SFs constant to 50 and vary the telemetry item requirements along the X-axis. We observe the linear growth, which is due to the linearly growing monitoring overhead for the P4 header pushing and parsing operations. Please note in Fig. 12 L represents Telemetry Frequency while in Fig. 10 and Fig. 11 it represents Telemetry Length (or telemetry items needed to be monitored).

We performed experiments with different substrate node topology from SNDLib to observe variance in the performance of the SARG approach with density of links (average degree of connectivity of nodes). For this purpose, we selected six different representative topology form SNDlib, as given in Table 5. Topology are chosen to represent different levels of degree of connectivity (DC) or the link densities.

Table 5
Topology from SNDLib.

Topology name	Nodes	Links	DC
Zib54	54	81	Low
Germany50	50	88	Low
Pioro40	40	89	Medium
Brain	161	332	Medium
Janos-US-CA	39	122	High
Giul39	39	172	High

Since placement of service flows is not the aim of this work, we assumed random placement of VNFs on substrate nodes. Accordingly physical links among substrate nodes are chosen by selecting one of many paths available to map service flow logical links on physical paths. No capacity calculations are performed as the placement is just symbolical. We varied total SFs from 50 to 100 with steps of 25. Results for total monitoring overhead instances and total time needed for execution of the algorithm are shown in Figs. 13 and 14 respectively. From the results, we observe that the monitoring overhead as well as the total execution time increases with total number of links in the topology as well as total number of SFs to be deployed. Since overheads (monitoring instances) for the two cases (Janos-US-CA and Giul39) are comparable, the total delays for 75 SFs are greater than that of 100 SFs and this fact can be attributed to the experimental and measurement errors.

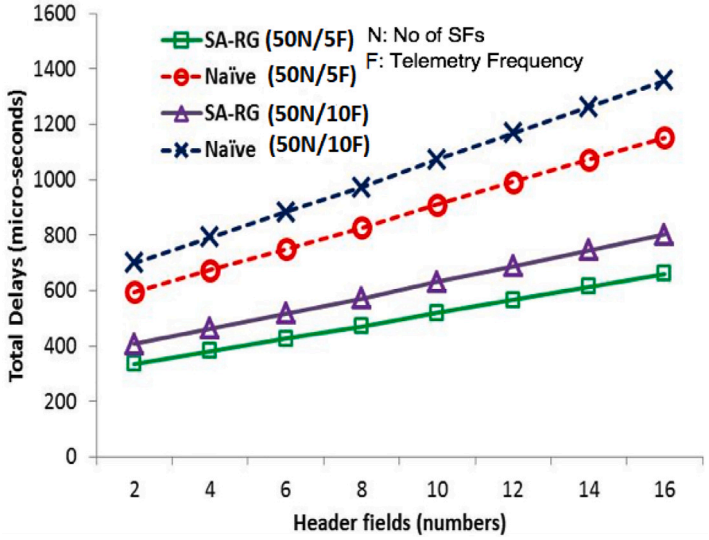


Fig. 12. Total delays against total telemetry items.

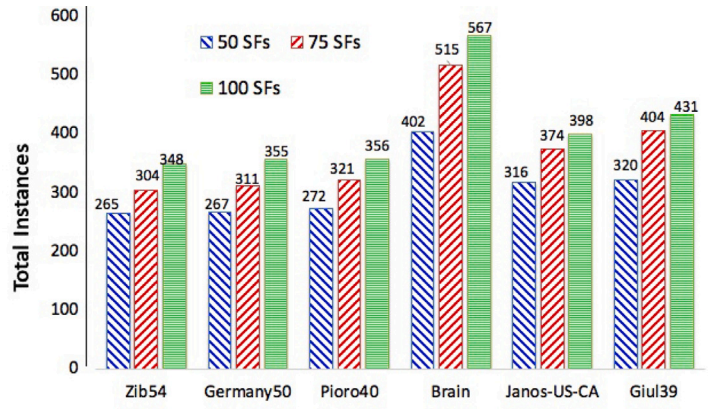


Fig. 13. Monitoring overhead for different topology from Table 5.

The node density or degree of connectivity has no effect on the total delays or overheads. This is expected, as our SARG heuristic considers only actually used physical links for the deployment of given SFs to deploy the MFs. For example, for Germany50, the total monitoring overhead with 100 SFs is 355 and that of Pioro40 is 356. Both topology have 88 and 89 number of physical links respectively. However, their node densities are low and medium. On contrary Janos-US-CA and Giul39 have same node density (high), however, they have 122 and 172 physical links. Hence their monitoring overheads differ significantly, that is 398 and 431 respectively. Total execution time of the heuristic also differs from 389.04 s to 437.93 s respectively.

We now observe the effect of initial temperature and cooling rate on the performance of SARG heuristic approach. We have selected Germany50 topology with 100 SFs to be monitored for this purpose. The plots of monitoring overhead and total execution time are shown in Figs. 15 and 16. From figures we observe that as the initial temperature in simulated annealing varies from 10^3 to 10^7, the performance improves. Similarly, the performance improves as the cooling rate drops from 10^{-3} to 10^{-5}. For example, Monitoring overhead in topology Germany50 for 100 SFs drops from 365 to 355 as the initial temperature and cooling rate values change from 10^3 and 10^{-3} to 10^5 and 10^{-5} respectively. The execution time of the algorithm increases from 1.5 s to 293.07 s. We also observe that performance improvement halts at the values of 10^5 and 10^{-5} for initial temperature and cooling rate. This might be due to the fact that either optimality has been achieved or the heuristic is stuck in the local minima, which is difficult to overcome. We stop the experiments at the values of 10^7 and 10^{-5} for initial temperature and cooling rate as the execution time increases significantly to 11676.34 s without any improvement in the solution quality.

Comparing the SARG approach to the näive approach, results presented in this section reveal a reduction of 39% in monitoring overhead (Encap/Decap instances) and a 57% reduction in overall monitoring delays (actual average packet delivery delays due to Encap/Decap operations). Numerical evaluation demonstrates that, with systematic approach such as SARG, monitoring overheads can be reduced significantly, which may enable existing monitoring platforms to be expressive as well as scalable. Please note that, our proposed model can easily extended for the differentiated QoS as well. For example, the SFC topology presented in Fig. 2 can easily be updated to add more nodes which can present entry and exit point of the switch and within these two nodes, different paths for different queues for differentiated QoS levels. This will result in larger Ξ matrix, as it represents topology for input SFCs. This may results in larger number of monitoring flows, as we may need one MF per tenant/queue. The effect of the larger number of SFCs as well as longer SFCs (more hops) on the total delays are already demonstrated in Fig. 12. In the next section, we present our concluding remarks.

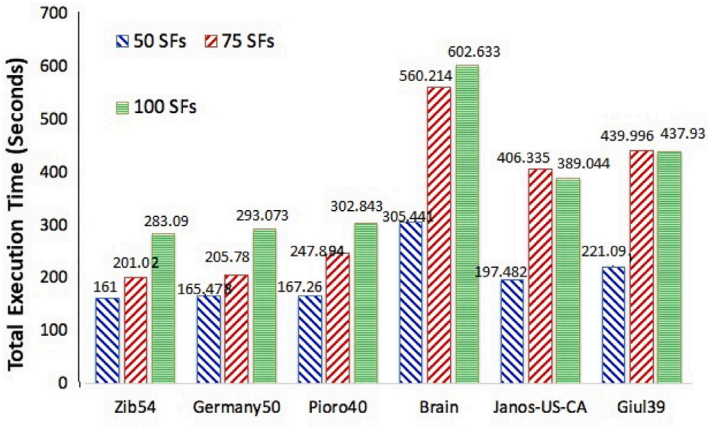


Fig. 14. SARG execution delays (seconds) for different topology from Table 5.

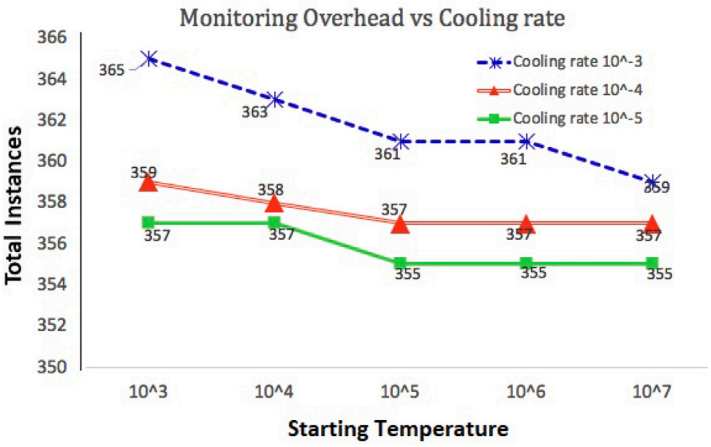


Fig. 15. Monitoring overhead vs. cooling rate for different starting temperatures.

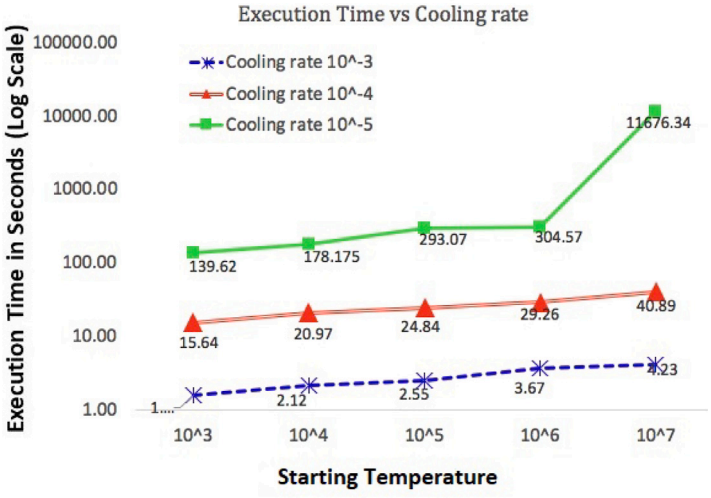


Fig. 16. Execution delays vs. cooling rate for different starting temperatures.

7. Conclusions

In this work, we propose IntOpt, a scalable and expressive telemetry framework designed for flexible VNF service chain network monitoring using active probing and P4. IntOpt allows to specify monitoring requirements for individual service flow in each network slice, which are mapped to telemetry item collection jobs that fetch the required telemetry items from P4 programmable data-plane elements. To quantify the gains achieved with the proposed framework, we develop mixed integer linear programming (MILP) model. We also implement simulated annealing based random greedy (SARG) meta-heuristic approach for near-optimal deployment of the monitoring flows (MFs) for flexible

Table 6
List of Acronyms.

Acronyms	Description
A&AI	Active and Adaptive Inventory
CPU	Central Processing Unit
DCAE	Data Collection, Analytics, and Event
DPI	Deep Packet Inspector
ePDG	Evolved Packet Data Gateway
FPGA	Field Programmable Gate Array
HSS	Home Subscriber Server
INT	In-band Network Telemetry
IoT	Internet of Things
IntOpt	In-band Network Telemetry Optimization
MF	Monitoring Flow
MIB	Management Information Base
MILP	Mixed Integer Linear Programming
MME	Mobility Management Entity
NFV	Network Function Virtualization
NS	Network Slice
NSP	Network Service Provider
OAM	Operations, Administration, and Maintenance
OF	Optimization Framework
ONAP	Open Networking Automation Platform
OSNT	Open Source Network Tester
QoS	Quality of Service
P4	Programming Protocol-Independent Packet Processors
SARG	Simulated Annealing based Random Greedy
SDC	Service De-sign Center
SDN	Software Defined Networking
SDNC	Software Defined Network Controller
SF	Service Flow
SFC	Service Function Chaining
SGW	Serving Gateway
SLA	Service Level Agreements
SO	Service Orchestrator
TOSCA	Topology and Orchestration Specification for Cloud Applications
VNF	Virtualized Network Function

service function chain monitoring. In addition to our proposed SARG meta-heuristic, we also implement an ad-hoc naïve approach, which is generally followed in the absence of a systematic flow generation strategy. We also benchmark monitoring overheads due to P4 telemetry operations for a single VNF N/W pipeline. Our evaluations demonstrate that using our proposed SARG heuristic, with proper initial temperature value and cooling rate, significantly reduces the total monitoring overheads by finding out near-optimal deployment of monitoring flows. This eventually reduces the delays and overheads introduced due to the telemetry operations. We argue that such systematic approach can be incorporated with the existing monitoring frameworks to obtain scalability without losing the generality and expressiveness of the systems. As a future work we aim to develop the integration APIs and make them available as an open-source for the integration of IntOpt with wide range of such network planning and automation tools. We also aim to provide the bench-marking results for other P4 platforms such as Netronome and others. For faster conversion, an approach based on constraint programming can be considered as an alternative to the proposed MILP in future.

CRediT authorship contribution statement

Deval Bhamare: Conceptualization, Methodology, Software, Writing – original draft. **Andreas Kassler:** Conceptualization, Supervision. **Jonathan Vestin:** Data curation. **Mohammad Ali Khoshkholghi:** Conceptualization, Methodology, Writing – review & editing. **Javid Taheri:** Conceptualization, Supervision. **Toktam Mahmoodi:** Conceptualization, Supervision. **Peter Öhlén:** Validation. **Calin Curescu:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

The authors received partial funding from the Knowledge Foundation of Sweden through the Profile HITS.

References

[1] D. Bhamare, A. Kassler, J. Vestin, M.A. Khoshkholghi, J. Taheri, IntOpt: In-band network telemetry optimization for NFV service chain monitoring, in: ICC 2019-2019 IEEE International Conference on Communications, ICC, IEEE, 2019, pp. 1–7.

[2] W. Ma, J. Beltran, Z. Pan, D. Pan, N. Pissinou, SDN-based traffic aware placement of NFV middleboxes, IEEE Trans. Netw. Serv. Manag. 14 (3) (2017) 528–542.

[3] L. Gupta, R. Jain, A. Erbad, D. Bhamare, The P-ART framework for placement of virtual network services in a multi-cloud environment, Comput. Commun. (2019).

[4] X. Li, M. Samaka, H.A. Chan, D. Bhamare, L. Gupta, C. Guo, R. Jain, Network slicing for 5G: Challenges and opportunities, IEEE Internet Comput. 21 (5) (2017) 20–27.

[5] X. Foukas, G. Patounas, A. Elmokashfi, M.K. Marina, Network slicing in 5G: Survey and challenges, IEEE Commun. Mag. 55 (5) (2017) 94–100.

[6] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, S. Uhlig, Elastic sketch: Adaptive and fast network-wide measurements, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, ACM, 2018, pp. 561–575.

[7] A. Fischer, D. Bhamare, A. Kassler, On the construction of optimal embedding problems for delay-sensitive service function chains, in: 2019 28th International Conference on Computer Communication and Networks, ICCCN, IEEE, 2019, pp. 1–10.

[8] J. Liu, W. Lu, F. Zhou, P. Lu, Z. Zhu, On dynamic service function chain deployment and readjustment, IEEE Trans. Netw. Serv. Manag. 14 (3) (2017) 543–553.

[9] L.-C. Wang, S.H. Cheng, Data-driven resource management for ultra-dense small cells: An affinity propagation clustering approach, IEEE Trans. Netw. Sci. Eng. (2018).

[10] J.G. Herrera, J.F. Botero, Resource allocation in NFV: A comprehensive survey, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 518–532.

[11] A. Gupta, R. Harrison, A. Pawar, R. Birkner, M. Canini, N. Feamster, J. Rexford, W. Willinger, Sonata: Query-driven network telemetry, 2017, arXiv preprint arXiv:1705.01049.

[12] M. Ashour, J. Wang, N.S. Aybat, C. Lagoa, H. Che, End-to-end distributed flow control for networks with nonconcave utilities, IEEE Trans. Netw. Sci. Eng. (2018).

[13] K. Borders, J. Springer, M. Burnside, Chimera: A declarative language for streaming network traffic analysis, in: USENIX Security Symposium, 2012, pp. 365–379.

[14] Y. Yuan, D. Lin, A. Mishra, S. Marwaha, R. Alur, B.T. Loo, Quantitative network monitoring with NetQRE, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, ACM, 2017, pp. 99–112.

[15] P. Lapukhov, R. Chang, Data-Plane Probe for In-Band Telemetry Collection, draft-lapukhov-dataplane-probe-01, Internet Engineering Task Force, 2016.

[16] R. Jain, S. Paul, Network virtualization and software defined networking for cloud computing: a survey, IEEE Commun. Mag. 51 (11) (2013) 24–31.

[17] H. Wang, R. Soulé, H.T. Dang, K.S. Lee, V. Shrivastav, N. Foster, H. Weatherspoon, P4FPGA: A rapid prototyping framework for P4, in: Proceedings of the Symposium on SDN Research, in: SOSR '17, ACM, New York, NY, USA, 2017, pp. 122–135.

[18] G. Tangari, D. Tuncer, M. Charalambides, Y. Qi, G. Pavlou, Self-adaptive decentralized monitoring in software-defined networks, IEEE Trans. Netw. Serv. Manag. 15 (4) (2018) 1277–1291.

[19] S. Lee, K. Levanti, H.S. Kim, Network monitoring: Present and future, Comput. Netw. 65 (2014) 84–98.

[20] Operation, Administration and Maintenance (OAM)Functions and Mechanisms for Ethernet Based Networks, G.8013/Y.1731, 2011, Available at: https://www.itu.int/rec/T-REC-Y.1731/en.

[21] IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management, IEEE Std 802.1ag - 2007 (Amendment to IEEE Std 802.1Q - 2005 As Amended By IEEE Std 802.1ad - 2005 and IEEE Std 802.1ak - 2007), 2007, pp. 1–260.

[22] L. Gupta, R. Jain, M. Samaka, A. Erbad, D. Bhamare, Performance evaluation of multi-cloud management and control systems, in: Recent Advances in Communications and Networking Technology (Formerly Recent Patents on Telecommunication), Vol. 5, (1) Bentham Science Publishers, 2016, pp. 9–18.
[23] A.G. Prieto, R. Stadler, A-GAP: An adaptive protocol for continuous network monitoring with accuracy objectives, IEEE Trans. Netw. Serv. Manag. 4 (1) (2007) 2–12.
[24] J. Case, M. Fedor, M. Schoffstall, J. Davin, A Simple Network Management Protocol (SNMP), RFC, (1157) 1990, pp. 1–35.
[25] A.S. Thyagaturu, Y. Dashti, M. Reisslein, SDN-based smart gateways (Sm-GWs) for multi-operator small cell network management, IEEE Trans. Netw. Serv. Manag. 13 (4) (2016) 740–753.
[26] T. Kohler, F. Dürr, K. Rothermel, Consistent network management for software-defined networking based multicast, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 447–461.
[27] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al., P4: Programming protocol-independent packet processors, ACM SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95.
[28] J. Vestin, A. Kassler, D. Bhamare, K.-J. Grinnemo, J.-O. Andersson, G. Pongracz, Programmable event detection for in-band network telemetry, 2019, arXiv preprint arXiv:1909.12101.
[29] R. Hark, D. Bhat, M. Zink, R. Steinmetz, A. Rizk, Preprocessing monitoring information on the SDN data-plane using P4, in: 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN, IEEE, 2019, pp. 1–6.
[30] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with OpenSketch, in: NSDI, Vol. 13, 2013, pp. 29–42.
[31] Q. Huang, X. Jin, P.P. Lee, R. Li, L. Tang, Y.-C. Chen, G. Zhang, Sketchvisor: Robust network measurement for software packet processing, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, ACM, 2017, pp. 113–126.
[32] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, V. Braverman, One sketch to rule them all: Rethinking network flow monitoring with univmon, in: Proceedings of the 2016 ACM SIGCOMM Conference, ACM, 2016, pp. 101–114.
[33] A. Kumar, M. Sung, J.J. Xu, J. Wang, Data streaming algorithms for efficient and accurate estimation of flow size distribution, in: ACM SIGMETRICS Performance Evaluation Review, Vol. 32, (1) ACM, 2004, pp. 177–188.
[34] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, C. Kim, Language-directed hardware design for network performance monitoring, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, ACM, 2017, pp. 85–98.
[35] H. Harkous, M. Jarschel, M. He, R. Priest, W. Kellerer, Towards understanding the performance of P4 programmable hardware, in: 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS, IEEE, 2019, pp. 1–6.
[36] D. Ding, M. Savi, G. Antichi, D. Siracusa, An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection, IEEE Trans. Netw. Serv. Manag. 17 (1) (2020) 75–88.
[37] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L.J. Wobker, In-band network telemetry via programmable dataplanes, in: ACM SIGCOMM, 2015.
[38] S. Tang, D. Li, B. Niu, J. Peng, Z. Zhu, Sel-INT: A runtime-programmable selective in-band network telemetry system, IEEE Trans. Netw. Serv. Manag. (2019).
[39] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, Flowsense: Monitoring network utilization with zero measurement cost, in: International Conference on Passive and Active Network Measurement, Springer, 2013, pp. 31–41.
[40] A. Lahmadi, A. Boeglin, Efficient distributed monitoring in 6lowpan networks, in: Proceedings of the 9th International Conference on Network and Service Management, CNSM 2013, IEEE, 2013, pp. 268–276.
[41] N. Alliance, Description of network slicing concept, in: NGMN 5G P, Vol. 1, 2016.
[42] S. Orlowski, M. Pióro, A. Tomaszewski, R. Wessäly, SNDlib 1.0–survivable network design library, in: Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium, 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009.
[43] Open network automation platform, 2019, Online; retrieved on Aug. 9th 2019, https://www.onap.org/.
[44] J.-P. Sheu, W.-T. Lin, G.-Y. Chang, Efficient TCAM rules distribution algorithms in software-defined networking, IEEE Trans. Netw. Serv. Manag. 15 (2) (2018) 854–865.
[45] D. Bhamare, M. Krishnamoorthy, A. Gumaste, Models and algorithms for centralized control planes to optimize control traffic overhead, Comput. Commun. 70 (2015) 68–78.
[46] Z.B. Zabinsky, Random search algorithms, in: Wiley Encyclopedia of Operations Research and Management Science, Wiley Online Library, 2010.
[47] H. Hamedmoghadam, M. Jalili, P. Moradi, X. Yu, A global optimization approach based on opinion formation in complex networks, IEEE Trans. Netw. Sci. Eng. (2018).
[48] MiniZinc, 2018, Online https://www.minizinc.org/. (Accessed 07 May 2019).
[49] IBM CPLEX, 2019, Online https://www.ibm.com/analytics/cplex-optimizer. (Accessed 07 May 2019).
[50] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, Exploring microservices for enhancing internet QoS, Trans. Emerg. Telecommun. Technol. 29 (11) (2018) e3445.

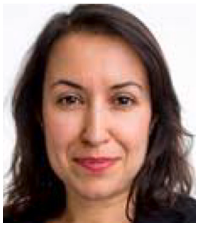
Deval Bhamare is working as an assistant professor in the University of Surrey, UK. He was working as a post-doctoral candidate at Karlstad University, Sweden, during development of this work. He has earned his dual Ph.D. from IITB-Monash Research Academy, 2015. His areas of research include Network Optimization, Middleware Architecture for Cloud Based Services, NFV-SDN, P4, In-band network telemetry and related topics.

Andreas Kassler is a Full Professor of Computer Science at Karlstad University, Sweden since 2005. From 2003 to 2004, he was Assistant Professor at the School of Computer Engineering, Nanyang Technological University, Singapore. He is co-chairing the Distributed Systems and Communication group. He is IEEE Senior Member and area editor for Elsevier Computer Networking Journal.

Jonathan Vestin received Ph.D. degree in computer science from Karlstad University in 2020. He is currently a Lecturer in computer science with Karlstad University. His research interest includes SDN and programmable networks, and their application for 5G networks.

Mohammad Ali Khoshkholghi received his Ph.D. degree in Computer Science from the University Putra Malaysia in 2017. He is currently a research associate in the Center for Telecommunication Research (CTR), Department of Engineering, King's College London (KCL), UK. Prior to joining KCL, he worked as a postdoctoral research fellow in the Department of Computer science, Karlstad University, Sweden. His research interests are Edge and Cloud Computing, Network Function Virtualization and Machine Learning.

Javid Taheri is a Full Professor at the Department of Computer Science at Karlstad University, Sweden. He received his Ph.D. in Mobile Computing from University of Sydney (Australia) in 2007. He is the recipient of the top 200 young researchers in the world by the Heidelberg Forum in 2013. His research interests include Cloud Computing, Edge/Fog Computing, Software-defined Networking, and AI-based optimization techniques.



Toktam Mahmoodi received her Ph.D. degree in telecommunications from King's College London, U.K. She was a Visiting Research Scientist with F5 Networks, San Jose, CA, USA, in 2013, a Postdoctoral Research Associate with the ISN Research Group, Electrical and Electronic Engineering Department, Imperial College, from 2010 to 2011. She is currently the Head of the Centre for Telecommunications Research, Department of Informatics, King's College London. Her research interests include 5G communications, network virtualization, and low latency networking.

Peter Öhlén is a principal researcher at Ericsson Research. In 2000 he received a Ph.D. in Photonics, also from the Royal Institute of Technology. He has been with Ericsson since 2005. He has worked with research and development in transport networks, network control, SDN, fiber access technologies, fiber-optic transmission, radio networks, optical and electronic subsystem design, simulation methods, project and program management.

Calin Curescu is a System Architect on Network Function Virtualization and Management with Ericsson. He is working on various topics, such as model-based orchestration, algorithms and optimization for allocation in the distributed cloud, multi-domain orchestration and management. He holds a Ph.D. in Computer Science from Linköping University, Sweden.